Cartographic documents for web modeling and representation of indoor mapping with interactive environments

[Extended Abstract] *

Ben Trovato†
Institute for Clarity in Documentation
1932 Wallamaloo Lane
Wallamaloo, New Zealand
trovato@corporation.com

G.K.M. Tobin‡
Institute for Clarity in Documentation
P.O. Box 1212
Dublin, Ohio 43017-6221
webmaster@marysville-ohio.com

Lars Thørväld§
The Thørväld Group
1 Thørväld Circle
Hekla, Iceland
larst@affiliation.org

Lawrence P. Leipuner
Brookhaven Laboratories
Brookhaven National Lab
P.O. Box 5000
lleipuner@researchlabs.org

Sean Fogarty
NASA Ames Research Center
Moffett Field
California 94035
fogartys@amesres.org

ABSTRACT

PUT THE ABSTRACT HERE

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory

Keywords

ACM proceedings, LaTeX, text tagging

1. INTRODUCTION

CARATTERISTICHE E SPUNTI:

- CONNESSIONE CON I SENSORI PER LA RILEVAZIONE DELLA POSIZIONE DEI VISITATORI
-
-

...

The remainder of this document is organized as follows. In Section II is provided an overview of the state of the art. Section III is devoted to describe the novel cartographic document proposed, while section IV introduces the underlying mathematical structure. Section V reports about the tools and instruments developed specifically for the web, focusing on software architecture, implemented algorithms and real applications. Section VI describes one use case of both document format and software tools. Finally Section VII proposes some conclusive remarks and future developments.

2. STATE OF THE ART

Researches about the cartographic representation of indoor environments are numerous and at the same time heterogeneos regarding to the strategies applied. Different sources of information are used and accuracy of the produced solution depends on the adopted approach. In some cases the information is obtained with automatic or semi-automatic processing of files that describe the architectural structure of a building, such as BIM (Link a BIM) and/or IFC (Link a IFC) [3]. Image processing is also used to extract topological information from floor plan images [4]. In other works, building informations and descriptive parameters are redefined from scratch [5]. This choice is driven by the unsuitableness of the current representive formats: images contain poor information and CAD files are not designed for this kind of use. A recurring topic among the use of carthograhic information is indoor navigation [4, 5, 3]. The proposed approaches are very different in this case too, and based on

*A full version of this paper is available as *Author's Guide to Preparing ACM SIG Proceedings Using LaTeX2ε and BibTeX* at `www.acm.org/eaddress.htm`

†Dr. Trovato insisted his name be first.

‡The secretary disavows any knowledge of this author's actions.

§This author is the one who did all the really hard work.

WOODSTOCK '97 El Paso, Texas USA

several strategies with some basic elements in common. An often adopted solution is based on the representation of the routing information as a graph, having a node for each room and an edge for each couple of connected rooms. In some cases the edges are weighted in function of euclidean distance. The detail level of the graph and hence the effective practicalbility of the calculated paths, can vary depending on the technique and the design choices applied, but in general most of the proposed solutions retrieve information only from architectual structure. A topic related to the navigation is the location of users. All the considered works agree on the inadequacy of GNSS in indoors contexts, due to the segnificant reduction in signal quality. To locate the exact position of a user inside a building, the currently most used techniques are based on fingerprinting and triangulation of radio signals (WiFi, Bluetooth, etc.) flanked by more original solutions based on, for example, image recognition [1]. User tracking issue is faced with solutions that range from the clever utilization of inertial tracking sensors embedded in many smartphones [1] to the adoption of ad hoc devices [4].

The actual "de-facto" standard in terms of geospatial data representation is the GeoJSON format, which can be easily used for any type of geographical annotation. In some cases it has been slightly adapted to be used in indoor environments (IndoorJSON).

2.1 The GeoJSON format

GeoJSON is a geospatial data interchange format based on JSON, suitable for a geometrical encoding of various geographic data structures. As opposed to GIS format, GeoJSON is an open standard. Positions need to be expressed in geographical coordinates (usually WGS84).

GeoJSON supports the following geometry primitives: `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString`, and `MultiPolygon`. Lists of geometries are represented by a `GeometryCollection`. Geometries with additional properties are `Feature` objects. Lists of `Feature` are represented by a `FeatureCollection`.

A single `Feature` is composed essentially by two mandatory fields: `geometry`, which describes the objects' geometry accordingly to the previously defined primitives, and `properties` which contains additional information about the `Feature`.

It is possible to define complex shapes through the composition of simple GeoJSON objects. Mainly due to its semplicity, GeoJSON is widely used and deeply integrated in several applications and services.

2.2 The IndoorJSON format

IndoorJSON is a GeoJSON variant defined and used by *indoor.io*, a Finnish company devoted to indoor environment mapping. IndoorJSON is compliant with GeoJSON syntax, and it may consist of any number of `Features` and/or `FeatureCollections`. All `Features` are interpreted similarly regardless of their grouping into nested `FeatureCollections`. IndoorJSON supports all GeoJSON geometry types.

Some particular properties are used to correctly define indoor elements:

- `level`: described which level contains the feature;
- `geomType`: identifies the object's category, useful during the visualization process.

There are some additional not mandatories properties useful for the indoor representation:

- `accessible`: describes if an element is walkable or not;
- `connector`: defines if the element is a connection between two levels;
- `direction`: describes the direction of the connection (both ways, only up, only down);

A syntax validator is provided by *indoor.io*, but the commercial nature of this project limits the number of tools available to deal with this format.

3. ADVANCES ON CARTOGRAPHIC DOCUMENT FORMATS

The focus of this work is the definition of a novel format of cartographic documents along with the software ecosystem rooted on it. A simple but effective algorithm to find indoor valid routes is also provided. The **HIJSON** (**H**ierarchical **I**ndoor **JSON**, this is the name chosen for the format) and the accompanying software framework aim to realize a mapping of indoor real spaces with a virtual interactive web environment. The HIJSON is based upon ideas and design principles collected from previous formats and identifies three critical improvements with respect to them: it exposes a *hierarchical structure*, uses *metric local coordinate system* and accepts *semantic extensions*.

3.1 Hierarchical structure

The HIJSON format allows for hierarchical description of indoor spaces. The introduction of a hierarchical structure establishes a parent-child relation between entities of the model, reflecting a container-contained relationship. This direclty implies a neater representation than the plain linear structure adopted by GeoJSON, being a perfect analogy of objects contained (i.e. placed) into spaces.

In addition, more organized arrangement is allowed by logical (or even physical) grouping: concepts like building wings, sections, stories, departemens, etc. can be introduced to reflect into the document structure logical or physical real divisions, categories or relashionships.

——————— LA PARTE SULLA TOPOLOGIA VA INTRODOTTA QUI E RIPRESA IN UNA SUCCESSIVA SEZIONE APPOSITA ———————

Hierarchical structures are common in computer graphics since they are used as scenegraphs. This accordance of underlying structures really simplify 3D render algorithms of HIJSON documented environments.

Furthermore the container-contained relation enables to use local reference system.

3.2 Metric local coordinate system

——- **revisionare** ———

Supported by the hierarchical underlying structure, the HJSON document format allows for the use of local coordinate system. This means that the shape of all elements can be conveninet modelled in local coordinate and than placed in the right position with respect to the position of the the parent (or container) element using a translation and a rotaction vector.

Another sostantial advantage is represented by the adoption of a metric reference, The adoption of a metric local reference system contribute to simplify the compilation of

the document, however it is, manual or aided by sofwtare tool.

Although both GeoJSON and IndoorJSON expresses all the positions in geographical coordinates can be useful for outdoor geographical representations, it is not the best solution for indoor descriptions.

Nevertheless countinuos outdoor-indoor navigation is guaranteed via ...

——- **fine revisione** ———

3.3 Semantic extensions

——- **revisionare** ———

Every HIJSON Element has a property that describes its class. This information allows the adoption of semantic extensions by the software that manipulates the HIJSON data. In the Javascript library developed to manage HIJSON documents, different classes are instantiated to represent HIJSON Nodes, which acts differently by the adoption of polymorphic methods. In order to extend the possibilities in representation and interaction, it is sufficient to define new classes that reflects new categories of HIJSON Elements.

——- **fine revisione** ———

4. HIJSON SYNTAX

Below there is an example of an input file ready to be processed by the HIJSON pipeline:

```
{
    "config": {
        ...
    },
    "data": [
                ...
                {
                    "id": "architecture",
                    "type": "FeatureCollection",
                    "features": [...]
                },
                {
                    "id": "furniture_1",
                    "type": "FeatureCollection",
                    "features": [...]
                },
                ...
    ]
}
```

The HIJSON document is composed of different parts:

- configuration: a JSON object containing parameters and settings useful for the building representation. In particular three points of the local reference system are mapped to three couples of geographical coordinates. This information allows the computation of the transformation matrix used to translate the local coordinates to global ones.
- one or more data collections: each of these lists is given in the form of a GeoJSON FeatureCollection, containing a number of HIJSON Elements. Since HIJSON Elements adhere to the GeoJSON format, each collection can be accepted by a GeoJSON validator. HIJSON introduces some additional rules that allow the adoption of this format for indoor representation. In the next paragraph is given a sample of HIJSON Element, with the description of the main differences from a standard GeoJSON Feature.

Below there is an example of data collection, with the definition of an HIJSON Element:

```
{
    "id": "architecture",
    "type": "FeatureCollection",
    "features":
    [
        ...
        {
            "type": "Feature",
            "id": "room_0.1",
            "geometry":
            {
                "type": "Polygon",
                "coordinates":
                [
                   [ [0, 0], [11, 0], [11, 19], [0, 19] ]
                ]
            },
            "properties":
            {
                "class": "room",
                "parent": "level_0",
                "description": "Office of Mr. Smith",
                "tVector": [10, 20, 0],
                "rVector": [0, 0, 90]
        },
        ...
    ]
}
```

4.1 HIJSON Element description

The first additional requisite above the GeoJSON format rules is the necessity of a unique ID, necessary for the referencing by possible child elements. The Geometry types allowed are `Point`, `LineString` and `Polygon`. Each geometry type is used to represent particular categories of elements (e.g. Polygons for levels and rooms, LineString for walls and doors, Point for furniture, etc.). The geometry coordinates are expressed in meters, and for convention starting at the bottom-left of the element. Unlike GeoJSON, where all the properties are optional, in HIJSON some attributes are mandatories:

- `class`: represent the element category, used to instantiate the appropriate semantic class;
- `parent`: contains parent's id of the nodes. The reason of the unique id depends on this property. The HIJSON Tree is created on the base of parent property;
- `tVector` and `rVector`: represent the translation and rotation relative to the parent element. The measure unit for translation is meter and for rotation is grades.

The definition of other properties is mandatory depending on the class of the element: For example the classes that defines internal or external walls require a `connections` array, containing the IDs of the adiacent areas. This information is used by the connector children of the element, like doors, to

identify the areas linked together. These connector elements are identified by a boolean `connector` property.

Optional fields can be added to improve the precision of the representation. Given the nature of the GeoJSON format from which HIJSON derives, the elements are represented by their 2D shape, like on a planimetry. To assign a value to the height of the object, intended as third dimension, the property `height` can be used.

A `description` property can provide further information about the element.

Additional optional fields can be added without restrictions, in order to enrich and extend the expressvity of the representation.

5. HIJSON TOOLKIT

The HIJSON Toolkit is a software module that implements common operations and transformations on HIJSON documents. Written in *JavaScript* language, it has been built to be deployed in the web environment. It is *modular* and entirely *isomorphic*, i.e. can run on the server as well as on every client. Working in the web environment, the Toolkit benefits of the fertility as regards the software development in this field: it takes advantage of libraries and frameworks such as *Ract*, "the JavaScript library for building user interfaces" by Facebook, and *Three.js* a framework to deal with *WebGL* tecnologies.

The Toolkit realizes the instantiation and extension logic of a HIJSON document, as described in the "HIJSON Class definition" section, and realize a multistage transformation pipeline that, as required, can be used entirely or only in part.

5.1 HIJSON processing pipeline

The HIJSON processing pipeline relizes the sequence of preliminary transformations that have to be applied to a HIJSON document before any futher operation. It is not strictly required to complete each stage of the pipeline: exit stage dipends on the specific use case.

The application of the transformation pipeline has a double aim. The first one consists in generating the graph of valid paths between all the interesting HIJSON elements. The second aim is the generation of one *GeoJSON* document for each story of the building described by the HIJSON document. In this way a bidimensional plant for each level of the building can be provided and visualized through any compliant GeoJSON viewer.

HIJSON processing pipeline (as pictured in figure AGGIUNGERE RIFERIMENTO) is composed by 6 elaboration stages. In the following are detailed operations excecuted by each stage, which are, in the order: *validation*, *georeferencing*, *parsing*, *graph paths generation*, *2D layers generation*, *marshalling*.

1. *`validation`* - The first one is the validation stage. In order to begin with the effective transformations the input HIJSON document must be compliant with the rules defined in (AGGIUNGERE REF TO PARAGRAFO REGOLE DI VALIDITA'). In the case the validation stage fails, processing aborts and do not continue to following stages. If the stage success, the output for the next stage is a validated HIJSON.
2. *`georeferencing`* - In the second stage, in order to allow for continuous outdoor/indoor navigation, the system needs to compute the georeferencing matrix, a linear operator able to transform local coordinates into global coordinates (referred to world coordinate system as latitude and longitude misures) and viceversa. This task is accomplished by solving a linear system obtained from information contained in HIJSON configuration part and precisely from the corrensponsdance of three real word points to three points included into the HIJSON document.
3. *`parsing`* - The parsing stage, takes the validated and georeferenced HIJSON as its input, that as illustrated before can be thought of as a list of HIJSON Elments, parses them and produce an istance of HIJSON Tree. The HIJSON Tree is an object in memory representing the tree hierarchical structure of the building described by the HIJSON document.
4. *`graph paths generation`* - The fourth stage is in charge of the generation of the graph paths. This aim is accomplished according to the algorithm described in (AGGIUNGERE RIFERIMENTO A #### Automatic generation of valid paths). The graph paths will be useful afterwards to coumpute valid paths from couple of point of interest on the graph. Once the graph paths has been computed, the input HIJSON Tree is augmented with paths information, becoming what has been called an HTAP (HIJSON Tree Augmented with Paths). Augmentation always takes place as leaf nodes added as children of a specific (e.g. "room") level.
5. *`2D layers generation`* - The fifth stage is the generation of GeoJSON layer. For each level, the system generates one geoJSON layer that will be use for the creation of 2D map. Each layer contains the children of 'level' node in the HIJSON Tree. Every class contains a boolean value that is used to choose which class will be a part of geoJSON layer. Every element has a geographical coordinates calculated by the transformation matrix with regard to the local coordinates of the HIJSON element.
6. *`marshalling`* - The last stage is responsible of execute

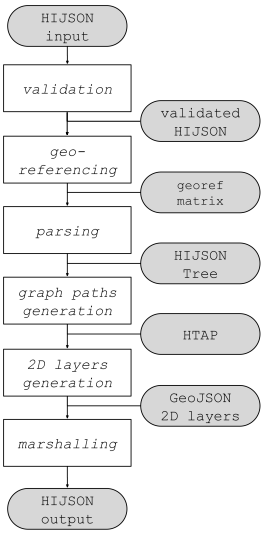


Figure 1: "HIJSON processing pipeline"

a serialization of the the transformed data. Tasks like breaking dependency-loops and stringification are performed. This stage is useful mainly serverside, as and the output is stored ready to be served to any requiring client.

5.1.1 Algorithmics: automatic generation of valid paths

The fourth stage of the processing pipeline is responsible for the generation of a graph of valid paths through the entire model represented by the intput HIJSON document. The graph generated according to the algorithm described in the following, although not optimal, ensures a complete coverage of the surface while limiting the numebr of generated nodes. Resulting graph is weigthed on the edges with nodes distances and each node represents alternatively:

a. standard path node, i.e. a junction node or possibly an endpoint of a path;
b. connection node, used as subproblem composing element in the divide et impera approch adopted (as described below).
c. element nodes ie. HIJSON Element (whose HIJSON Class explicitly grants his presence in the graph), typically an endpoint of a path;

Such a graph allows for directions calculations between any two given nodes. Although different approaches have been explored [2], a very classical solution has been selected in this case, so directions are actually computed clientside applying the Dijkstra's shortes route algorithm on the graph.

Taking advantage of the hierarchical structure of the HIJSON document, and according to the divide et impera approach, the problem of the graph paths generation is splitted in several sub-problems which consist in the computation of the sub-graphs relative to each room, or more generally ambience. The sub-graphs are then linked together through the connection nodes (which in most cases represents doors). The resolution of each sub-problem (as depicted in figure METTERE RIFERIMENTO ALLA FIGURA), is composed by 4 phases:

1. Computation of the walkable area of the ambience: this task is accomplished subtracting area of the possibly encumbrances to the area of the ambience; the result is tipically a surface with holes;
2. Triangulation of the walkable area: the computed surface is triangulated taking into account the presence of holes;
3. Identification of graph nodes: for each triangle side completely internal to the area, its midpoint is selected as standard path node;
4. Junction of nodes: nodes relative to the same triangle are then linked together; both element nodes and connection nodes (i.e. doors) are linked to the nearest node in the ambience (i.e. room).

5.2 HIJSON Class definition

To exploit the possibilities offered by HIJSON Toolkit, along with the HIJSON document, some custom dynamic behaviours must be described. These behaviours encapsulate the specificities relative to comminucation procols with the sensors as well as user interaction peculirities. The interface for these behaviours is the HIJSON Class.

Figure 3: "HIJSON Element/Class/Node relashionship"

Each HIJSON Element of the HJSON document given as input, has a dynamic counterpart, a running instance called HIJSON Node, instantiated according to the corresponding HIJSON Class via relfection methods (see picture XXXXX).

To specify a new HIJSON Class means to extend the Toolkit to deal with a new class of HIJSON Element.

To extend the toolkit to deal with a new class of HIJSON Element is required to to specify a new HIJSON Class, defining the following properties and methods:

- `in_graph`: a boolean value to express if the element is an approachable point in the graph paths;
- `in_2D_map`: a boolean value to express if the element is wanted in the 2D map;
- `get2DStyle`: a method that returns the 2D map appearence of the element, essentially HTML and CSS code;
- `get3DModel`: a method that returns the 3D model appearence of the element, a *THREE.js Object3D*;
- `getWidget`: a method that returns the information widget, a *React* component;
- `getProxy`: a method that returns server side proxy which encapsulate IoT sensor communication protocol, a *Node.js module*.

User's needs for new indoor elements, different sensor equipment, alternative representation on 2D or 3D viewport are accepted by the definition of new HIJSON Classes that allows in this way single point custom extension of the Toolkit capabilities.

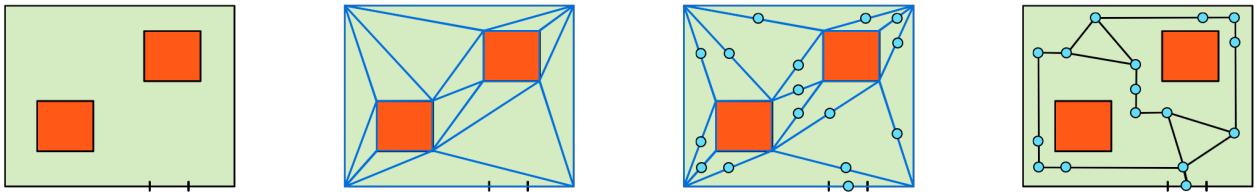
6. HJSON WEB FRAMEWORK

The HJSON Web Framework responds to the needs of an extendable, customizable, and scalable framework which provides at the same time IoT monitoring, realtime multiperson tracking and crossfloor user navigation.

Expandability and customizability derives from both design choises and HIJSON inherent characteristics, the possibility of semantic extensions. Scalablility is directly borrowed from technologies used for the software development: *JavaScript* language, using *Node.js*, in particular *Express.js* as backend framework, exploiting the power of WebSocket protocol through the *Socket.io* library.

Being supported by the web as bearing platform, the framework exposes also an highly availability: it is so simple to use as to visit a website, both from desktop or mobile devices, without explicit requirements to install any software from proprietary stores (access to which is often denied from business devices).

The HIJSON Web Framework deeply relies on HIJSON Toolkit and offers the overal client/server architecture and a convinient, highly intractive user interface, leaving aside



(a) (b) (c) (d)

Figure 2: Graph paths generation: (a) detection of obstacles and computation of walkable area; (b) triangulation of walkable area; (c) identification of graph nodes area; (d) junction of nodes.

the specific indoor positioning system and the IoT sensors, to deal with a robust interface is provided and described in the following seciton.

6.1 Applications

The Framework has been designed focus on two different possible kind of users: the *Explorer* and the *Supervisor*. They have different requirements and are likely equipped with different devices: while the *Supervisor* monitors the indoor environment through a desktop workstation, the *Explorer* has a smartphone available and needs to be routed across the building.

6.1.1 IoT monitoring

Every element in the HIJSON environment is capable of showing information about itself, so it can be analyzed by the user. The modularity of the HIJSON Toolkit permits to show particular information or UI about a specific object, using polymorphic behaviours of the different HIJSON Nodes. If an object is connected to the network and it is capable of interaction, the user can benefit of its functions through the system (e.g. if the object is a thermostat, the user can see the temperature in the room and can turn on/off the heating). If the object isn't interactive, the system can show static information (e.g. for fire Extinguisher, the system shows the last date of checking).

6.1.2 Realtime multi-person tracking

A typical task performed by a *Supervisor* can be the monitoring of users locations inside the building. This operation can be required for various reasons, e.g. security or logistics. The devices that equips the *Explorers* can be used to track their position in realtime, giving the *Supervisors* the whole picture of the presences inside the building in every moment.

— da controllare cosa tenere rispetto a quanto giÃă scritto dopo —

The system can be used for access monitoring. On 2D map will be a marker for each person in the building, whereas on the 3D model will be a 3D model of user. With an appropriate system of indoor localization, every person sends its position in real time. The 2D map and the 3D model is automatically refreshed. The user can ask for information about person, in function of the use of the system.

The communication of the current position happens with the socket.io library. Every user sends an object like this:

```
currentPosition = {
    coordinates: [x, y],
    levelId: level-ID
}
```

For every change of the coordinates or level, an *emit* event will be generated, according to the socket.io library, which sends to the server the new current position. The server sends to the connected client the updated current position of the users connected. In the client-side, when the *on* event of socket.io library is hit, there will be a refresh of the marker in 2D map and the 3D model in the 3D representation. With Observer pattern implemented in socket.io library, it is possibile to have a realtime multi-person tracking. This system permits to be indipendent from the tool that gets the position of the users: the only requisite is to send the position like the object described above.

— fine controllo —

6.1.3 Crossfloor user navigation

As shown in the algorithmics section, the HIJSON Toolkit provides a particular strategy to assemlbe a graph of possible paths inside the building. This graph, represented also in the form of a weighted adjacency matrix, can be easely used to compute paths between two nodes inside the building. To achieve this result, the matrix, which edges are weighted according to the distance between two nodes, is used as input in an applicaiton of the Dijkstra's algorithm. The result is the shortest path between two selected nodes of the graph. Thanks to the crossfloor connections of nodes representing stairs or elevators, the paths calculated can also start and end on different stories.

6.2 Architecture

Like the vast majority of the web based application, the Framework exposes an overall architecture that is inherently *client/server*. In particular, two different type of possible client are identifiable: the *Supervisor* client and the *Explorer* client. Both of them connect to the same server.

The indoor space described by the HIJSON document provided as input is processed by the server (cfr. PROCESSING PIPELINE). After that any connecting *Explorer* client, presumably through a mobile device, will be provided with the information to perform crossfloor navigation of the building, while reporting user position to the server. The server will feed any connecting *Supervisor* client with users positions, along with data from sensor-equipped things present in the environment, realizing the IoT monitoring and the realtime multi-person tracking.

6.2.1 Server Architecture

The architecture of the server is depicted in XXXX. A

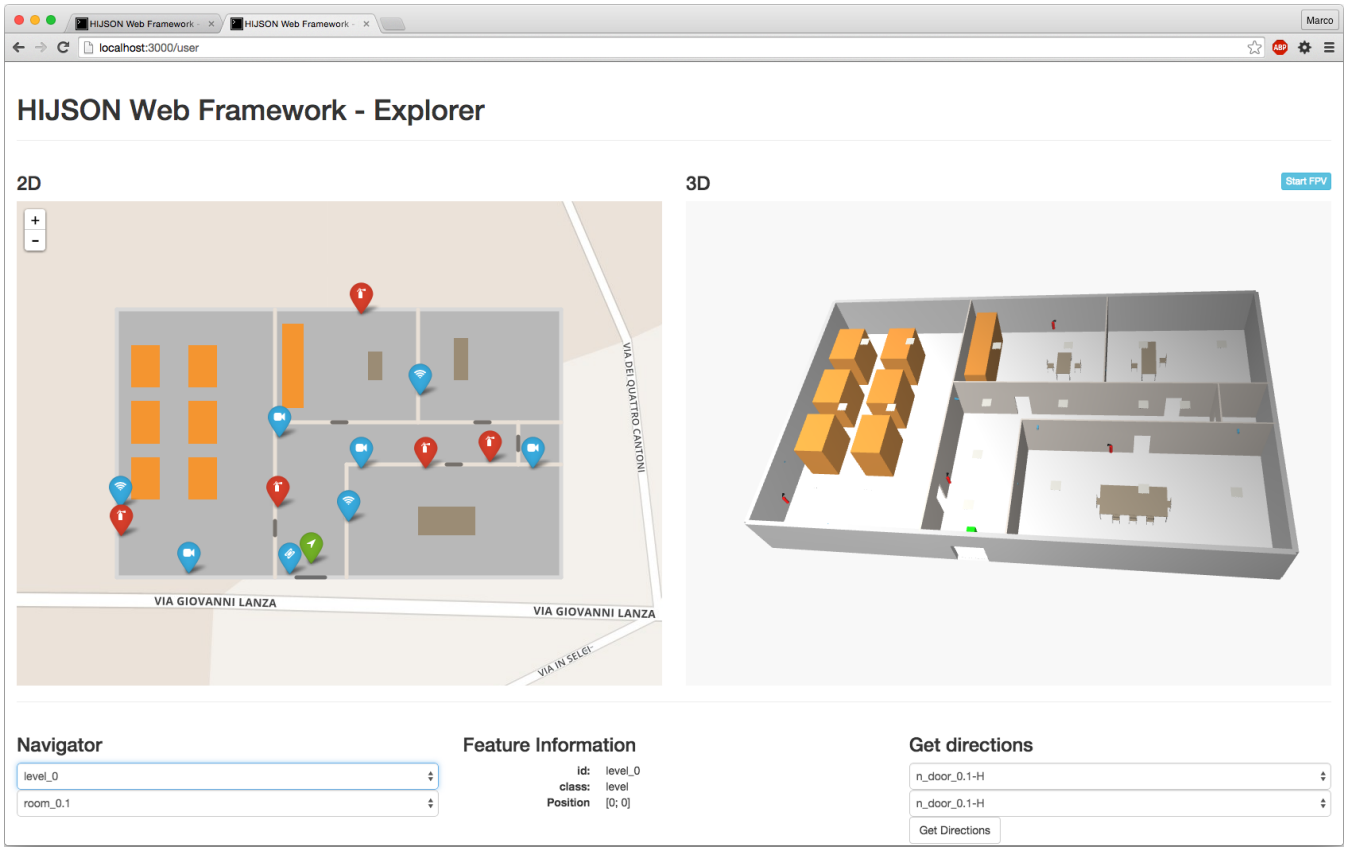


Figure 4: "HIJSON Web Framework UI"

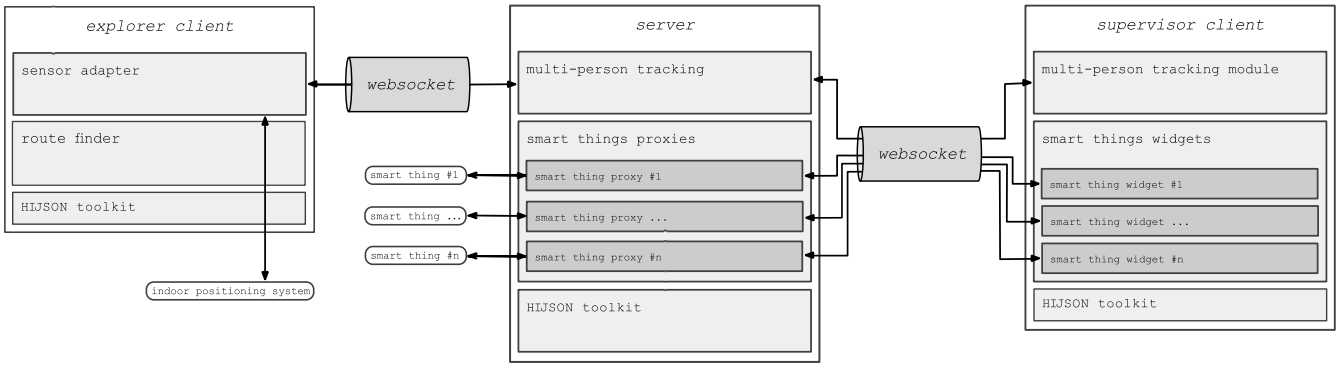


Figure 5: "HIJSON Web Toolkit architecture"

web server module is responsible for listenning to connecting clients. Each client connection is handled by the web server module providing all the required resources and opening one websocket channel, through which will flow *Explorer* and/or *Supervisor* communication protocol data. In particular, `multi-person tracking` module receives position data from *Explorer* clients. It aggregates and sends these information to connected *Supervisor* clients through the websocket channel, using a simple but realible protocol later described. Indipendence from particular IoT sensor equipment communication protocol is achieved by the `smart things proxies` module through the proxies modules obtained by the HIJSON Class definition (`getProxy` method previuosly described).

6.2.2 *Explorer client architecture*

The *Explorer* client architecture is generally deployed on a mobile device, usually supplied to a user who needs to be routed across the environment described by HIJSON document. The `sensor adapter` module encapsulates the communication logic with the indoor positioning system. The presence of this module ensures indipendece from particularly tecnology allowing client *Explorer* to rely on different indoor positioning systems: INDICARE ESEMPI SENSATI E CUTTING EDGE DI POSIZIONAMENTO INDOOR. INTRODURRE IL PROBLEMA DELLA COMU-

NICAZIONE A LIVELLO API DEL BROWSER CON SENSORISTICA PER LA RILEVAZIONE DELLA POSIZIONE.

Every time the `sensor adapter` observe a perceptible modification in user position sends the new position information to the server through the single opened websocket, using the a message with the following syntax:

```
currentPosition = {
    coordinates: [x, y],
    levelId: level-ID
}
```

Relevant information includes, beside current coordinates, the indication of the story of the possibly multilevel building the user is in. The `smart things widgets` module being in common with the client *Supervisor*, has been treated in the next section.

6.2.3 Supervisor client architecture

The Client *Supervisor* architecture shows two modules. The first one, the `multi-person tracking` module, is responsible to receive through the websocket, from the server information about explorers of the environment, showing them in the user interface. The second module, the `smart things widget` communicates with the server to propose to the user realtime information about sensor-equipped objects in the environment. Data passes through the single websocket opened between the server and every *Supervisor* Client. Rely on a naive but effective communication protocol, each smart thing widget exchaange data only with respective smart thing proxy on the server. To ensure the the data is sent only when the user requires the information relative to a specific smart thing, a widget lifecycle protocol is implemented: it is based on the 4 events `on_before_show`, `on_show`, `on_before_hide`, `on_hide` triggered, as suggested by their names when a widget is shown or hidden. When the user requires information about a smart thing, its widget has to be rendered, but `on_before_show` the server is notified to connect via relative proxy to the sensor. Once connected, the server begin to send data via webscocket. Received data is shown through the wodget to the user. When done, or `on_before_hide` event of the widget, a notification is sent to the server announcing to stop sending data and the proxy close the connection to the sensor. Widget lifecycle protocol ensures that only requiring data is sent from th eserver to the client.

7. CONCLUSIONS

We presented HIJSON a GeoJSON extension for indoor mapping TRA GLI SVILUPPI FUTURI: - GENERAZIONE GRAFICA IN AMBIENTE CAD DEL DOCUMENTO HIJSON

8. REFERENCES

[1] B. Al Delail, L. Weruaga, M. Zemerly, and J. Ng. Indoor localization and navigation using smartphones augmented reality and inertial tracking. In *Electronics, Circuits, and Systems (ICECS), 2013 IEEE 20th International Conference on*, pages 929–932, Dec 2013.

[2] W. Bian, Y. Guo, and Q. Qiu. Research on personalized indoor routing algorithm. In *Distributed Computing and Applications to Business, Engineering and Science (DCABES), 2014 13th International Symposium on*, pages 275–277, Nov 2014.

[3] M. Boysen, C. de Haas, H. Lu, X. Xie, and A. Pilvinyte. Constructing indoor navigation systems from digital building information. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 1194–1197, March 2014.

[4] L. Faramondi, F. Inderst, S. Panzieri, and F. Pascucci. Hybrid map building for personal indoor navigation systems. In *Advanced Intelligent Mechatronics (AIM), 2014 IEEE/ASME International Conference on*, pages 646–651, July 2014.

[5] D. Gotlib, M. Gnat, and J. Marciniak. The research on cartographical indoor presentation and indoor route modeling for navigation applications. In *Indoor Positioning and Indoor Navigation (IPIN), 2012 International Conference on*, pages 1–7, Nov 2012.